Organization of the No. 1 ESS Stored Program

By J. A. HARR, MRS. E. S. HOOVER and R. B. SMITH

(Manuscript received January 22, 1964)

The stored program of the No. 1 ESS must perform switching functions reliably and promptly. Its design must be economical of memory and execution time, and must accommodate office growth easily. The program is organized so that an interrupt system initiates the input-output programs that must be performed to accurate timing tolerances. The data collected by these input-output programs are passed to the call processing programs which decide what course of action the calls shall follow. The program also assigns an appropriate share of central control time for maintenance and administrative functions. The program is generic in the sense that specific quantities which change from office to office are looked up in tables and are not embedded in the program itself. This article describes the basic program structure and illustrates it with some typical programs.

I. INTRODUCTION

The program for the No. 1 electronic switching system (ESS)[1] constitutes the operating intelligence of the system. In fulfilling this function it must meet the stringent operating requirements of a telephone switching office. This article describes the most important of these requirements and outlines the implementation selected as a result.

There are six main classes of requirements. First, the system must respond appropriately in real time to demands for service. Second, the system must perform a large variety of actions to provide the many services which are offered and to work compatibly with a wide range of connecting systems. Third, the system must be extremely reliable. In forty years of continuous operation, the total time the central processor may be out of service is measured in minutes. Fourth, the system must work in wire centers which are growing in the amounts of equipment and in the scope of features offered. Fifth, although there will be a large number of installations which may differ from each other in the kind

and quantities of equipment and in the services which they offer, costs of compiling programs must be kept at a minimum. Finally, the program should be designed to keep system costs low whenever possible, principally the cost of program storage, the cost of call storage, and the cost of the required number of central processors needed for the total market.

1.1 *Requirements on Processing Capability*

Telephone customers are used to receiving prompt service whenever they request it. As a result, much of the call processing work performed by a switching system cannot be postponed for long. For example, when a customer answers a ringing telephone, ringing stops in less than a quarter of a second and the two customers may begin conversation.

A particular wire center also receives signals from other wire centers as well as from customers directly. The size and extent of the total investment in switching equipment makes it necessary for new equipment to communicate with older systems. For example, because step-by-step switching trains are driven by the customer's own dial pulses, the step-by-step office will spill digits into another office without checking to see if the receiving office is ready. Therefore, if calls are to be processed correctly, the No. 1 ESS must be ready to receive dial pulses from a step-by-step office in a few hundredths of a second from the time the trunk is seized.

A third source of demand for prompt action comes from the character of the switching system which the program must control. The relay circuitry has been much simplified compared to that in previous common control systems. However, relays are still used both in the network controllers and in trunk, junctor, and service circuits. Simplification has been possible because the program has taken on the job of triggering the operation of relays in proper timing and sequence. In the case of the dial pulse receiver, for example, pulse detection, counting, and memory functions are performed by the program. In order to make sure that the pulses are detected under the worst cases of pulse distortion, the program must look every 11.5 milliseconds for a change in the line current. The higher the repetition rate of scanning, the larger the percentage of time which the central processor must spend in scanning. Because the duration of the scanning program itself varies with the number of pulses detected, there is variation in the actual rate of scanning a given receiver. Study has shown that when a group of receivers is scheduled to be examined every 10 milliseconds, the interval between inspections of any particular receiver will almost never exceed

11.5 milliseconds. Thus, certain functions are performed by the program on fairly tight timing tolerances in order to simplify circuitry in the rest of the switching system. The need to perform a number of tasks with different tolerances is inherent in the nature of the switching function. Failure to organize the program properly would result in inefficient use of the central processor, which in turn would mean a larger investment in data processing equipment.

1.2 *Versatility*

A fundamental reason for using a stored program in the No. 1 ESS is the need for versatility. The No. 1 ESS must be compatible with other switching equipment and must provide the ever growing list of special services which the telephone industry offers to its customers. By using a stored program, these objectives can be attained economically. Even though much development work may go into designing programs for new features, the program, once written, can be easily installed. Only the contents of memory must be changed and not a large number of wired connections. Because the program must perform so many functions, it is very large. To keep it manageable, it is divided into functional blocks which are utilized in different ways to perform a large number of complicated tasks. When a new service is needed, much of the required program may be already available. The manner of dividing the program is discussed in the paper on call processing.[2]

1.3 *Reliability*

The ability to process calls is the principal function of the central processor and its program. Of almost equal importance is the ability to do so reliably. A complete failure of a central office for even a period of 15 minutes is an event so rare that it generally is reported in newspapers. Yet the No. 1 ESS has so centralized the intelligence of the central office that no call can proceed without the attention of the central processor. Therefore, when a failure or malfunction is detected in the central processor, the faulty unit must be switched out and a working system put together from the duplicated units. The faulty unit must then be diagnosed so that plant personnel may quickly repair it and put it back into service.

About half of the total program instructions are devoted to fault recognition, diagnosis, and routine maintenance. The detection of a fault, the analysis of which unit is faulty, and the actions taken to assemble a working central processor are assigned the highest levels of

priority. Once a working system has been put together, the detailed diagnosis of the faulty unit can proceed at a more leisurely pace than that of normal call processing.

Included within the call processing programs are checks for abnormal inputs and processing errors. For example, a customer ought not to be able to generate more than ten dial pulses before there is a pause while the dial is wound up again. If, however, there is a weak power cross which has been undetected by the power cross test, a longer series of dial pulses may be generated. The input program which counts pulses checks for an excessive count and treats such a call as a partial dial.

1.4 *Ability to Accommodate Growth*

A given wire center will have a unique set of engineered equipment. With growth, the amount of equipment changes. It would be expensive to rewrite the program every time an additional frame of equipment is added to the center. Therefore the basic program is designed to treat all information about quantities of office equipment as data which can be changed as the office grows.

1.5 *Standardization of the Program*

Additional flexibility is required of the program to meet the needs of wire centers with differing features. One wire center needs to communicate with step-by-step switching equipment, another with panel switching equipment, a third offers TOUCH-TONE calling service to its customers, and so on. Producing a tailor-made program for each office would increase programming and compiling costs and entail problems in guaranteeing an error-free program. Therefore, a further requirement is that a single program contain all features needed for a wide range of applications. This standard program, called a *generic* program, may thus be used in many installations.

1.6 *Economic Balance*

The length of the program in program store, the number of call store words, and the number of machine cycles required to perform a task can all be traded off against each other. These quantities have associated costs; for example, the number of machine cycles affects the total amount of processing equipment required to serve a given market. Therefore an over-all requirement is achievement of a sound economic balance in the use of these quantities.

The following pages describe the organization and implementation of the program plan chosen to meet these requirements.

II. PROGRAM STRUCTURE

The organization of the No. 1 ESS program is strongly influenced by the fact that it must operate in real time. That is, the program must respond promptly to signals and data submitted to it by customers and other switching systems. In addition, it must respond quickly to errors detected by the many trouble detector circuits designed into the hardware to assure dependable operations within the system at all times. Whenever it fails to do so, the result may be improper handling of calls and a general degradation of service. For example, failure to detect digital signals may result in directing a call to a wrong number, or failure to outpulse digits to another office promptly will cause the other office to return overflow tone to the calling customer. Therefore it is necessary to establish a hierarchy of program tasks. Some tasks must be performed on a strict schedule; others may be delayed without significant adverse effects.

2.1 *Interrupt System*

The central processor has an interrupt mechanism[3] within it which seizes control of the system momentarily when a manual, trouble detector, or clock signal is received. The interrupt circuit causes the system to stop its present program task, store the program address at which it was interrupted, and then transfer to the appropriate fault-recognition program or clock-controlled input-output program. When the interrupt programs are completed, control is returned to the program that was interrupted.

Fig. 1 illustrates this over-all program plan. The interrupt sources and their associated programs are arranged in a hierarchy of nine interrupt levels; from highest to lowest, these levels are designated A, B, C, D, E, F, G, H, J. An interrupt source assigned to a particular level can interrupt programs of lower levels only. The majority of the programs are subject to interruption by any of the nine levels, and are therefore called *base-level* programs.

The highest interrupt source is the A level, initiated from the master control center, which allows manual selection of operating modes. Interrupt levels B through G are activated by system trouble detectors. These fault-recognition programs are discussed in the article describing maintenance[4] appearing in this issue.

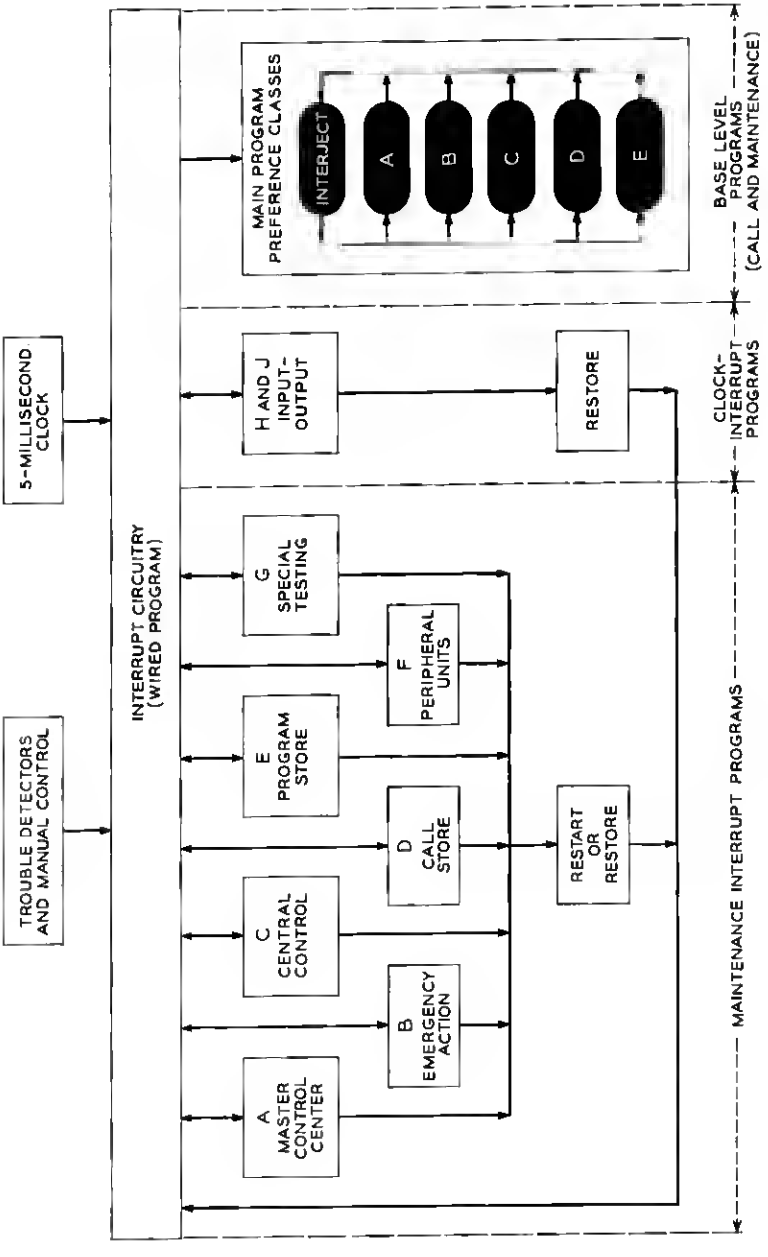

Fig. 1 — Program control plan.

Every 5 milliseconds a system clock activates interrupt level J, which in turn gives control to input-output programs. Level H is used to interrupt the level-J input-output program when tasks being performed exceed 5 milliseconds. The level-J programs are normally in control for about 0.2 to 2 milliseconds, according to the size and traffic of the office.

2.2 *Input-Output Buffering*

In order to go to the next task reasonably promptly, the individual tasks must not take too long. In particular, it is necessary to limit the amount of processing performed by interrupt-level programs. For example, the scan of the lead on a TOUCH-TONE receiver that indicates that a digit is present ends by reading the ferrods associated with the individual frequencies and placing the result in an area of call store known as a "hopper." It would be possible to carry the processing of the call further. The area in call store where the other digits dialed on this call are stored could be interrogated to determine whether dialing is finished, and if so, the network path hunt and ensuing actions required to set up ringing could be carried out. However, all this work would consume several milliseconds of continuous processing for a single call. Since the scan for a signal present on a TOUCH-TONE receiver must be performed every 10 milliseconds, an excursion of several milliseconds each time a digit is detected would frequently exceed tolerances. Hence, work on the call is terminated by buffering the digit in a hopper, from which a base-level program will later unload it and carry out the processing just described.

Fig. 2 is a schematic representation of the program control and flow which carries out the sequence of actions needed to process calls, described elsewhere in this issue.[2]

The input-output programs are shown in the upper part of this figure. The input programs are confined to scanning for and recognizing input signals and storing this information in a call store hopper. The hopper stores the input information until the base-level programs inspect it for data. When data are present, appropriate base-level programs, as shown at the bottom of Fig. 2, start or continue the processing of the call. Likewise, call store buffers are provided for the base-level programs to load with output data. At an appropriate time, these data are unloaded by an output program which delivers them to the peripheral equipment. Some examples of the hoppers and buffers used in the system are shown in the middle of Fig. 2. For example, the peripheral order buffer (POB) is used to store address and control data for network controllers and

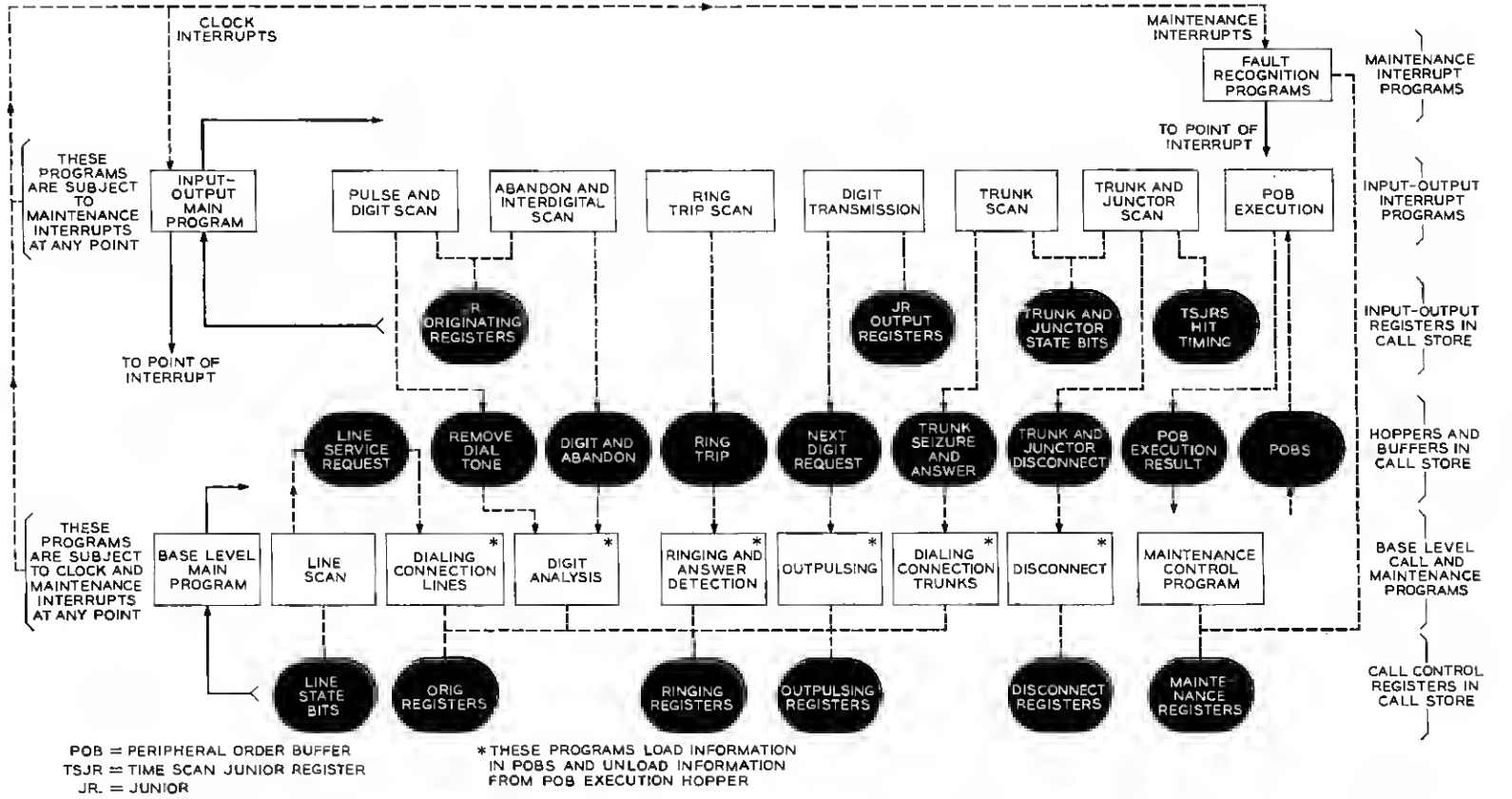

Fig. 2 — Schematic representation of program control and flow.

signal distributors. These buffers provide the means for communication between the scheduled input-output programs and the base-level call processing programs.

III. PROGRAMS PERFORMED ON THE CLOCK INTERRUPT LEVEL

The 5.5-microsecond clock pulses in the central control are counted, and every 5 (actually 5.005) milliseconds the counting circuit generates an output signal which interrupts the base-level program being performed. The interrupt circuit makes the central processor transfer to the J-level input-output main program. Some input-output tasks are performed every 5 milliseconds; others are performed at multiples of 5 milliseconds up to 120 milliseconds. The input-output programs are classified into high-priority and low-priority tasks, according to the frequency and accuracy with which they must be performed.

The low-priority tasks can be delayed for a few milliseconds without adverse effect on the operation of the system. This indeed will be the case when the coincidence of input work under a peak traffic load causes the system to take more than 5 milliseconds to complete the high- and low-priority tasks. In this event the H-level interrupt will occur and the low-priority work will be interrupted. The accumulated high-priority work will again be performed before returning to the low-priority program that was interrupted.

Accordingly, each input-output program is assigned to either the high- or low-priority timetable. A list of high-priority tasks, the frequency at which they must be executed, and the call store memory words needed for carrying out these tasks are shown in Table I. A similar list of low-priority tasks is shown in Table II.

3.1 *Input-Output Main Program*

To assist in understanding the design of the input-output main program, a description of some additional program attributes is needed.

TABLE I — HIGH-PRIORITY INPUT-OUTPUT TASKS

Task	Frequency	Call Store Memory
(1) Dial pulse and digit scan	10 ms	junior originating registers, remove dial tone and digit hoppers
(2) Abandon and inter-digital timeout	120 ms	junior originating registers, digit and permanent signal partial dial hoppers
(3) Twistor card writing	5 ms	twistor word storage register
(4) Call charge magnetic tape output	5 ms	message storage registers

TABLE II — LOW-PRIORITY INPUT-OUTPUT TASKS

Task	Frequency	Call Store Memory
(1) Teletypewriter scan	25 ms	teletypewriter buffer
(2) Peripheral order	25 ms	peripheral order buffer (POB), POB execution hopper
(3) Power cross test scan	*	POB execution hopper
(4) Ringing current test scan	†	POB execution hopper
(5) Detection of outgoing trunk wink	100 ms	outpulsing junior register, next-digit request hopper
(6) Multifrequency outpulsing	25 ms	outpulsing junior register, next-digit request hopper
(7) Disconnect scans	10 ms	timed-scan junior registers, trunk and junctor disconnect hopper
(8) Interrupt sanity test	100 ms	emergency-action register
(9) Interject timing	100 ms	interjected ordered bits buffer

* This task is performed on three consecutive 5-ms intervals after a power cross scan order is encountered during a POB execution.

† This task is performed on the first and third of the five 5-ms intervals after a ringing current test scan order is encountered during a POB execution.

3.1.1 *Characteristics of the Input-Output Main Program*

Since this program must be executed every 5 milliseconds, the time required by the central control to cycle through all of the input-output task programs is held to a minimum, even at the expense of a small increase in the total number of program words. For example, it is expedient in some cases to have a number of program blocks that perform nearly equal tasks instead of a common program capable of performing all of the tasks, since the common program would in general involve more machine operations to accommodate the small variations in each of the individual tasks.

The program plan is sufficiently flexible that the same program can provide service after changes in the system due to growth. Also, the same program must operate during and after certain changes in the features offered by an office. For example, a feature included in the generic program may require a type of trunk circuit which is not provided in all offices. These circuits may be introduced into any office without changing the program. To meet this growth requirement, the information relating to size, traffic, and features for an office is not embedded in the data field of program instructions, but instead is provided in parameter tables within the program store. In cases where the real time of the system is quite sensitive to the access time for retrieving a particular parameter, it is also stored in the call store, from which it can be read more quickly than from the program store. In particular,

two parameters of this type are the number of input-output tasks and the frequency with which they are performed.

3.1.2 *Organization of the Input-Output Main Program*

A block diagram of the input-output main program is shown in Fig. 3. When the J-level interrupt occurs, control is transferred to the input-output main program, which operates as follows:

(1) It saves the contents of the central control index registers to allow resumption of the base-level program at the point of interruption and sets the H-level inhibit flip-flop.

(2) It updates the time counter and activates, one at a time, all input-output programs that require action according to the high-

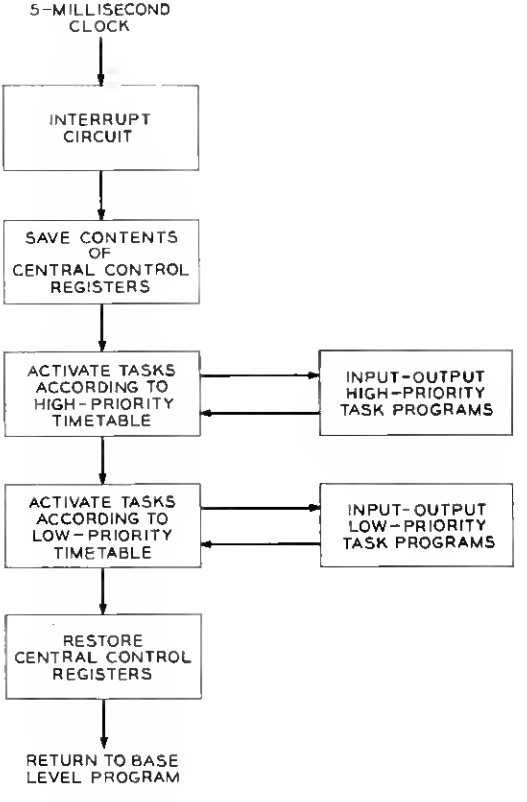

Fig. 3 — Input-output main program.

priority timetable. When all high-priority tasks are completed, it uninhibits the H-level interrupt by resetting the control flip-flop.

(3) It activates, one at a time, all input-output programs that require action according to the low-priority timetable.

(4) It then refills the central control index registers with the information saved in (1) and returns control to the interrupted base-level program.

The principal parts of the input-output main program are the high- and low-priority timetable programs. Since both programs are identical in structure, it is necessary to examine only one of them to understand the operation of the input-output main program.

3.1.3 *High-Priority Timetable Program*

Figs. 4 and 5 show the layout of the call store used to support the high-priority timetable program. The transfer table of Fig. 4 consists of 23 consecutive words. Here and in the following descriptions, symbolic designations will be used — in this case, P0 to P22 — instead of

P	0	DIAL PULSE AND DIGIT SCAN PROGRAM ADDRESS 1
	1	DIAL PULSE AND DIGIT SCAN PROGRAM ADDRESS 2
	2	
	3	
	4	
	5	
	6	ABANDON INTERDIGITAL TIMING SCAN PROGRAM ADDRESS
	7	
	8	
	9	CALL CHARGE PROGRAM ADDRESS
	10	
	11	
	12	
	13	
	14	
	15	TWISTOR CARD WRITING PROGRAM ADDRESS
	16	
	17	
	18	
	19	
	20	
	21	
P	22	ZERO CNT PROGRAM ADDRESS

Fig. 4 — Call store transfer table for high-priority timetable program.

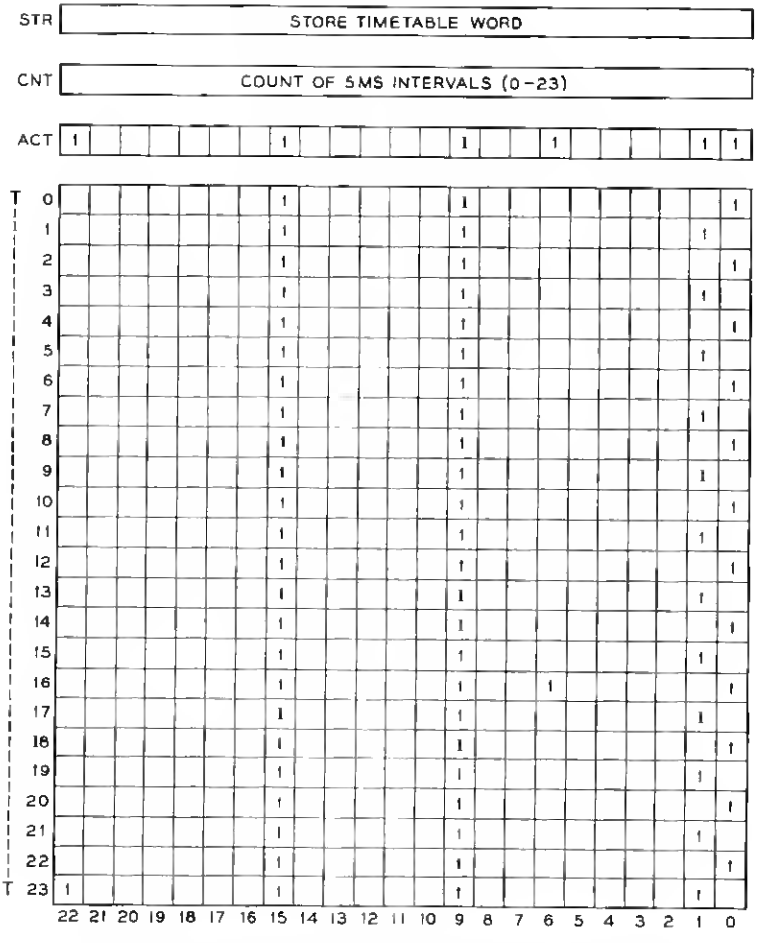

Fig. 5 — Call store timetable for high-priority timetable program.

numerical call store addresses. Each of the 23 words is used to store the starting address of an input-output program assigned to the high-priority timetable. For instance, words P0 and P1 contain the program store addresses of the dial pulse and digit scan programs.

The timetable of Fig. 5 consists of 24 consecutive words designated T0 to T23. Each word is associated with a particular 5-millisecond interval within a 120-millisecond cycle. A count kept in the CNT word is used to identify the 5-millisecond interval and the associated T word in the timetable.

Within each T word, bits 0 to 22 are associated with the input-output

programs whose starting addresses are stored in the corresponding words P0 through P22. Thus each column of the timetable is associated with an input-output program. Each column has an activity bit which is part of the ACT word. If the activity bit of a column is equal to 1, the 1's marked within the column designate the 5-millisecond intervals during which the associated input-output program is due for execution. The table shows that program P0 is activated on all even 5-millisecond intervals and that program P1 is activated on all odd 5-millisecond intervals. Program P15, when active, is due every 5 milliseconds. Program P22 is activated only once every 120 milliseconds. This program recycles the count in CNT from 23 to 0.

The word STR is used to store the pattern of 1's and 0's that indicate which input-output programs are due for execution during an interval. As these programs are executed, one at a time, the corresponding flag bits are reset to 0 until all the programs have been executed.

A step-by-step description of the program shown in Fig. 6 is given in Table III. Assume that the count of 5-millisecond intervals that is kept in CNT is initially 0.

As shown in Fig. 6, to activate the first input-output program requires eight program steps and subsequent ones only five. Note that only six of the possible 23 flag columns are presently used in this high-

LOCATION	OPERATION	ADDRESS OR DATA, REGISTER AND OPTION FIELDS
ENTER	MX	CNT
	MK	TO, XA
	XM	CNT
	PMK	ACT
	TZRFZ	LP
	KM	STR
	MY	PO, F
	T	O, Y → TO INPUT-OUTPUT TASK PROGRAM
LOOP	MK	STR ← FROM COMPLETED TASK
	TZRFZ	LP → TO LOW PRIORITY TIMETABLE PROGRAM
	KM	STR
	MY	PO, F
	T	O, Y → TO INPUT-OUTPUT TASK PROGRAM

Fig. 6 — High-priority timetable program.

TABLE III — STEP-BY-STEP DESCRIPTION OF HIGH-PRIORITY TIMETABLE PROGRAM

(1)	MX	CNT	The content of CNT is read into central control (CC) register X. (This value of the interval count is used as a pointer to select one of the 24 timetable entries.)
(2)	MK	T0,XA	The address T0 is indexed with the value of the X register to obtain the address of the timetable entry to be read into CC register K. Then the content of X is increased by 1. (In successive interrupts, this instruction results in reading the timetable entry at address T0+0, T0+1, T0+2, and so on up to T0+23.)
(3)	XM	CNT	The new value of the interval count is stored in CNT.
(4)	PMK	ACT	The timetable entry contained in K is ANDed with the activity bits in ACT; the resulting word is placed in register K. This word contains a 1 in every position in which both the timetable entry and the ACT word contain a 1. Thus, each position marked by a 1 designates a program that is active and due for execution. For instance, when T0 is read, three bits are equal to 1 in columns 0, 9, and 15.
(5)	TZRFZ	LP	If all the bits in K are 0 (if no programs need to be acted on), the program transfers to address LP where the low-priority timetable program starts. If one or more bits of K are equal to 1, the position of the rightmost 1 is stored in register F. The rightmost 1 itself is set to 0, and the program advances to the next step. Clearing the rightmost 1 allows the next 1, if any, to be recognized later as the new rightmost bit of the word in K.
(6)	KM	STR	The new binary word in register K, modified by the removal of the rightmost 1, is stored in STR.
(7)	MY	P0,F	The address P0 is indexed with the value in register F to obtain the transfer table entry to be read into register Y. This is the address of the input-output program corresponding to the rightmost 1, originally in register K.
(8)	T	0,Y	The program transfers unconditionally to the address stored in register Y as the result of step (7). When the input-output program has been completed, a transfer is made back to step (9).
(9)	MK	STR	The word stored in STR is read into register K.
(10)	TZRFZ	LP	If all the bits of the word in K are 0, the program transfers to the low-priority timetable program. If one or more bits are equal to 1, the position of the new rightmost 1 is stored in register F, that 1 is erased, and the program advances to the next step.
(11)	KM	STR	This step performs the same function as step (6); it stores any remaining flag bits in STR.
(12)	MY	P0,F	This step performs the same function as step (7); it obtains the address of the next input-output program to be executed and stores it in Y.
(13)	T	0,Y	A transfer is made to the address in register Y. When the input-output program has been completed, a transfer is made to step (9). Steps (9) to (13) are used over and over until, one by one, the flag bits are removed. The TZRFZ instruction of step (10) will then lead to the low-priority timetable program.

priority timetable. Therefore ample space for future input-output programs is provided. All that has to be done to add programs is to place appropriate time flag bits in any one of the unused columns, mark the column activity bit to 1, and place the corresponding entry address of the new input-output program in the transfer table.

This flexibility has been gained at the cost of only 17 additional call store words in the transfer table. By writing appropriate data into the call store (and a copy in the program store for reliability), an input-output program included within the generic program can be added to an office, and changes can be made in the order and frequency of execution of any input-output program. Furthermore, the input-output main program need not be changed even if a new issue of a generic program requires the addition of new programs.

An example of a specific input-output high-priority task program is given in Section 3.2.

3.2 *Dial Pulse and Digit Scan Program*

This program is used to scan the signal-present leads of dial pulse, multifrequency, and TOUCH-TONE receivers for signals every 10 milliseconds. It is activated during every 5-millisecond interval by the high-priority timetable program. However, only half the receivers are scanned in each of the 5-millisecond intervals in order to even out the work load. The functions of this program are as follows:

(1) For dial pulse receivers this program detects pulses by observing a change in the scanner reading from 0 to 1 (off-hook to on-hook condition of the handset). When such a change is found, the program adds one to the pulse count, which is located in a call store area called a "junior register" associated with each receiver. If this is the first pulse received, the address of the originating register in which digits are to be accumulated for this call is read out of the junior register and loaded into the remove dial tone hopper. In case the pulse count overflows (becomes equal to 16) the program stops incrementing the pulse count so that the permanent signal partial dial program will detect a timeout and make an entry in the permanent signal partial dial hopper.

(2) For TOUCH-TONE receivers this program detects that TOUCH-TONE signals are present by observing a change in the signal-present scan point reading from 0 to 1. Upon finding a change, the tone frequency scan points are read and their values together with the address of the originating register serving the call are stored in a TOUCH-TONE digit hopper.

(3) For multifrequency receivers this program detects the presence of multifrequency signals by observing a change in the signal-present scan point value from 0 to 1. The scanner leads associated with each frequency are read, and their values together with the address of the originating register serving the call are stored in the multifrequency digit hopper.

Observe the similarity of the program actions required to detect and report inputs from the three types of receivers. This design permits flexibility in the assignment of scan points for all three types of receiver, since the input program scans a row of scanner points for a change in reading from 0 to 1 without regard to the type of receiver being interrogated.

It is beyond the scope of this paper to discuss the complete dial pulse and digit scan program. However, a description of the core of this program reveals its basic design.

Fig. 7 shows the call store memory layout used by the core program. Each word in the scanner address table (SCA) contains a trunk scanner number of a row of digit receivers. This number addresses a word of 16 scanner outputs, among which at least one is assigned to a receiver.

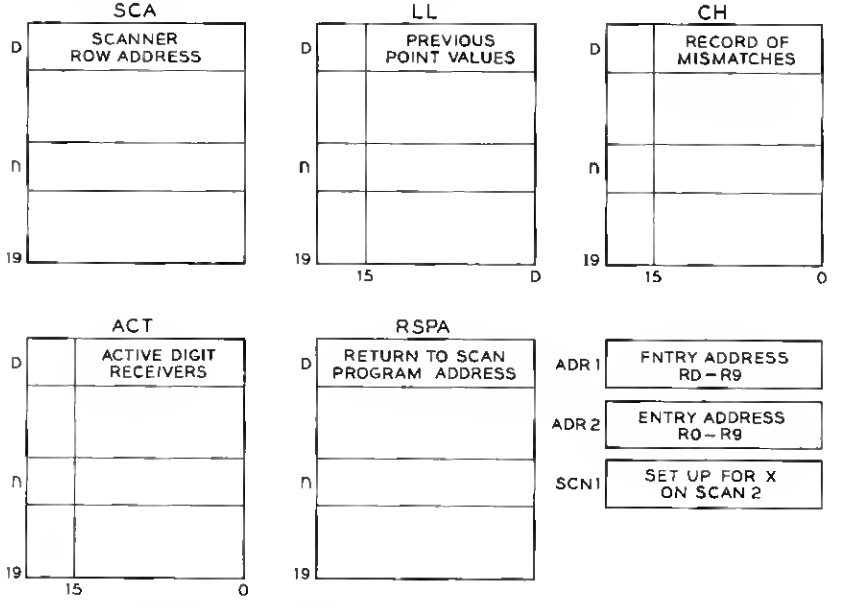

Fig. 7 — Call store memory layout for core of scan program.

The last look table (LL) stores the previous scanner outputs of the row in a memory location corresponding to the location of the row address in the scanner address table.

The change status table (CH) is used to record a mismatch between the present and previous scanner readings when detected. The corresponding change status bit for the scanner is set equal to 1. The abandon and interdigital timing scan uses this information to determine interdigital timeout or abandonment of the call by a customer.

The activity table (ACT) specifies the active receivers within each scanner row and is used by the dial pulse or digit present program to determine the active receivers reporting a change in state from 0 to 1.

The return to scan program address table (RSPA) contains the appropriate return address for use by the dial pulse or digit present program to return control to the core program after the pulse counter(s) is (are) incremented and/or the appropriate hopper(s) is (are) loaded.

Observe that all of the tables described above are blocks of 20 words each. This design was made to accommodate the maximum number of receivers required in a wide range of offices. The symbolic name of the call store address of the first word of each table is SCA, LL, CH, ACT or RSPA, respectively. The binary addresses corresponding to these names will be defined by the compiler-assembler[5] in the fixed area of the call store. That is, these tables have been assigned a fixed location in call store in all central office programs. However, for small offices only a small number of rows in each table will be used to accommodate the receivers for the office. Therefore, the core program is designed to scan only the number of rows required in a given office. Since approximately half the number of rows are to be scanned in each of the 5-millisecond intervals, three other control words are needed by the core program.

At a call store location with the symbolic name ADR1 is stored one of the entry addresses R0 through R9 according to the number of rows to be scanned during the even 5-millisecond intervals. Likewise, at address ADR2 is stored the appropriate entry address for the number of rows scanned during the odd 5-millisecond intervals. At address SCN1 is stored a number one less than the number of rows scanned in the even interval. This number is used in the odd interval scans to set an index register to the proper value for controlling the row words read and stored in memory.

The functions performed by the core program shown in Table IV are:

(1) Initialize central control registers according to whether the scan occurs in an even or odd interval.

(2) Read the row of 16 scan points represented by the first scanner

TABLE IV — DIAL PULSE AND DIGIT SCAN CORE PROGRAM

Location	Operation	Variable Fields	Comments
SCAN1	WX	−1	This is the entry point on even 5-ms intervals. Set index register X to −1.
	MSF	SCA+1,X	Send scanner address in first word of SCA table to scanners. Present state of 16 scan points returns to central control register L.
	WZ	DPDP	Store in central control register Z the address (DPDP) of entry point to dial pulse or digit present program.
	MK	CH+1,X	Move content of first CH word into central control register K.
	T	ADR1,M	Transfer program control to the address in call store location ADR1; this will be one of the addresses R0 to R9, depending on the number of rows to be scanned in this office.
SCAN2	MX	SCN1	This is the entry point on odd 5-ms intervals. Set index register to one less the number of rows scanned in the even interval.
	MSF	SCA+1,X	(Same as SCAN1+1)
	WZ	DPDP	(Same as SCAN1+2)
	MK	CH+1,X	(Same as SCAN1+3)
	T	ADR2,M	(Same as SCAN1+4, except that call store address ADR2 is used instead of ADR1.)
R0	UMKMJ	LL+1,X	R0 is the entry point from SCAN1+4 or SCAN2+4 if ten rows are to be scanned. Match (exclusive-OR) present scan point values in register L with previous scan point values read from corresponding word in LL table. Store in central control register J the 16-bit mismatch word (1's at bit positions of mismatch), and OR it with the CH word in register K, storing the result in K.
	LM	LL+1,X	Move present scan point values from register L into the LL table.
	KM	CH+1,XA	Move into table CH from K the new CH word, which now contains 1's in bit positions corresponding to scan points whose state changed since the last abandon-interdigital scan. Also add one to register X.
	JKMSF	SCA+1,X,PL	AND the mismatch word in J with the present scan point word in L and store in K this logical product, which now contains 1's only in the bit positions where a 0-to-1 change occurred in the corresponding scan points. Set a decision-control flip-flop, to be used by the following order, according to whether the product in K is zero or not. Also read the next scanner address from the SCA table and send it to the scanners. The scan point values will return to L in two machine cycles.
	TAUMK	CH+1,X	If the product in K is nonzero, transfer program control to the address in central control register Z. If zero, move into K the contents of the next word in CH table.

The next 45 steps essentially repeat the preceding five instructions nine more times to take care of a possible ten rows of scan points to be examined. The last few instructions in the tenth set of five are modified slightly because it is not necessary to prepare for a next row in that case. A concluding instruction transfers control back to address LOOP in the high-priority timetable program (Fig. 6).

row address in the SCA table if it lies in the even interval, or scanner row address one greater than the number stored in SCN1 if it lies in the odd interval.

(3) Match the 16 new scan point values with the previous scan point values stored in the corresponding row of 16 LL bits.

(4) Update the corresponding row of 16 CH bits. That is, write 1's into the CH word wherever a mismatch (change in scan point state) occurred.

(5) Update the row of LL bits. That is, read present scan point values to the corresponding row in the LL table.

(6) Determine those scan points whose value changed from 0 to 1.

(7) Transfer to the dial pulse or digit-present program if there is at least one change from 0 to 1.

(8) Repeat steps (2) through (7) for the remaining scanner rows to be examined during this 5-millisecond interval.

(9) Return program control to the high-priority timetable program.

Because of the frequency of use of this core program, an economic balance clearly required an emphasis on machine cycle minimization at some cost in total memory. Two means were used to achieve this real time efficiency. First, three special instructions (UMKMJ, JKMSF, and TAUMK) were designed, each of which accomplishes functions that would require two or three general instructions. Second, the five-word core program is repeated ten times in the program store to avoid the additional time for executing loop control orders and transferring. The program is generic, in that it is designed to handle traffic over the range of office sizes without appreciable loss of efficiency. This flexibility is attained in exchange for a moderate increase in the total call store and program store words used.

IV. BASE-LEVEL PROGRAMS

The bulk of No. 1 ESS programs, both call processing and maintenance, are executed on the base level — that is, with none of the interrupts A through J in effect. An appreciable amount of time is spent in the J level, as described above, looking for inputs and disposing of outputs, but this time is consumed in repeated execution of a relatively small number of fairly short programs. The complex work called for by the enormous variety of call situations and equipment configurations and possible malfunctions is carried out by a correspondingly large number of programs. These are executed as required, when time permits, and while no interrupts are in effect. All base-level programs

can be deferred to some extent, but the amount of delay they can tolerate varies widely. It is for this reason that a preference system and a program organization to implement it are required within the base level.

4.1 *Main Program*

This plan and its associated programs are referred to as the base-level main program, or simply the "main program." The question of precisely which associated programs should be considered a part of the main program is a rather arbitrary matter of classification. The fact is that the entire collection of base-level programs becomes in a real sense a *single* program, though subdivisions are useful for various purposes such as functional design and explanation, assignment of work to individual programmers, convenience of assembly and testing, and so forth.

Maintenance programs on the base level, as well as call processing programs, receive control from the same main program. They follow the same general rules and share many devices and techniques that are discussed below. However, since the structure of the maintenance programs is described in detail elsewhere,[4] details and examples in this paper are drawn primarily from the processing of telephone traffic.

Just as the bulk of No. 1 ESS programs are base-level, so the bulk of base-level programs are "task" programs. These are simply the programs at the end of the line of control that perform the ultimate work of the office. Most do a particular kind of work on a single call at a time; others perform various administrative functions, such as interpreting a teletypewriter input message. They differ, too, in the manner in which they receive control from the main program. Some receive it directly from a single dispenser program, while others require a series of dispensers, each triggering the next.

Within the main program complex are distinguished several kinds of "dispenser" programs. These form the links between the basic schedule of the main program and the task programs. One of the principal jobs of the dispenser programs is unpacking the input data that the H and J interrupt-level programs have buffered and distributing them to the task programs for analysis and use. Another important dispenser function is providing timed entries to programs requiring them.

All base-level work is divided into six classes. The highest priority class, called "interject," is described below. The other five are the classes A, B, C, D, and E, in descending order of frequency of examina-

tion. Specifically, the main program delivers control to these classes according to the pattern

ABACABADABACABAEABACABADABACAB

repeated endlessly. Thus class A is examined most frequently, and tasks assigned to that class will suffer shorter delays on the average than those in B, and so on. But if, for example, class E work is about to be done, the existence of waiting class A work cannot change the scheduled order of execution.

Each class consists simply of a number of dispenser programs that are executed in a fixed order, so that if $a_1, a_2, \cdots, a_5$ represent the five dispenser programs in class A, $b_1, b_2, \cdots, b_5$ the five in class B, and $c_1, c_2, \cdots, c_6$ the six in class C, the sequence of execution stated above can be expanded to:

$$a_1a_2a_3a_4a_5b_1b_2b_3b_4b_5a_1a_2a_3a_4a_5c_1c_2c_3c_4c_5c_6a_1a_2 \cdots.$$

4.2 *Dispenser Programs*

As an illustration, class B consists of the dispenser programs that administer work from the following five buffers:

class B ordered bits buffer
ring trip hopper
remove dial tone hopper
dial pulse and abandon hopper
trunk and junctor disconnect hopper.

"Buffer" is used as the general term for any memory used to store or record work to be carried further at a later time; a "hopper" is a buffer used to accumulate inputs from peripheral equipment.

The ring trip and remove dial tone hoppers are typical single-purpose hoppers. The nature of the input, in each case, is known automatically by the interrupt-level program that detects it, so it is just as easy for it to be loaded into its own hopper, and more efficient for the unloading dispenser program. The remove dial tone hopper is shown in Fig. 8. The "pointer block" of four words contains both a loading and an unloading address, so that first-in, first-out service can be given. Also, it contains the addresses of the beginning and end of the actual hopper, so that the loading and unloading programs are completely independent of the hopper's size and location. The information that is entered in this hopper requires only one word, and consists merely of the call store address of the originating register whose call is ready to have dial tone

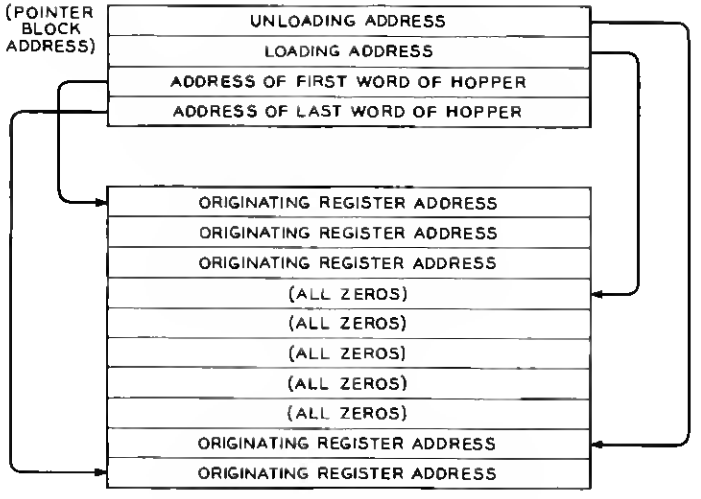

Fig. 8 — Release dial tone hopper.

removed. (A request to remove dial tone is made after detection of the first pulse, so that it will actually be removed by the time the first digit is dialed.) Many hoppers are of this same form, except for different kinds and quantities of information in each entry; some require two words per entry.

Once a dispenser program receives control, it empties its buffer of all work it finds there. Thus, if the remove dial tone dispenser finds the five entries shown in Fig. 8, it will pass control five times to the same task program, each time with central control index register X containing one of the originating register call store addresses unloaded from the hopper.

The dial pulse digit and abandon hopper (whose entry format is shown in Fig. 9) has a dual purpose. An interrupt scan program detects both types of input by the same timing; the only difference between

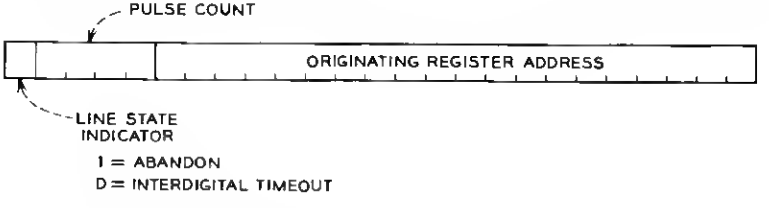

Fig. 9 — Format of entry in dial pulse digit and abandon hopper.

them is that the line is off-hook if a digit has been completed, but on-hook if the call has been abandoned. It is more efficient, therefore, for the interrupt-level program to put both types into the same hopper, with one bit of the entry indicating on-hook or off-hook, than to make the decision itself and place them in separate hoppers.

Each priority class has an ordered bits buffer; the one for class B is shown in Fig. 10. Such a buffer consists of a call store word, each bit of which serves as a flag for its associated program. The address of this program is in an accompanying table in a position corresponding to the flag's bit position in the word. Thus, in Fig. 10, the word BOBB contains two 1's. The rightmost 1, in bit position 3 (starting with 0 on the right), indicates there is work to be done by the program whose address is in word number 3 of the table BP0. This program, as seen from the

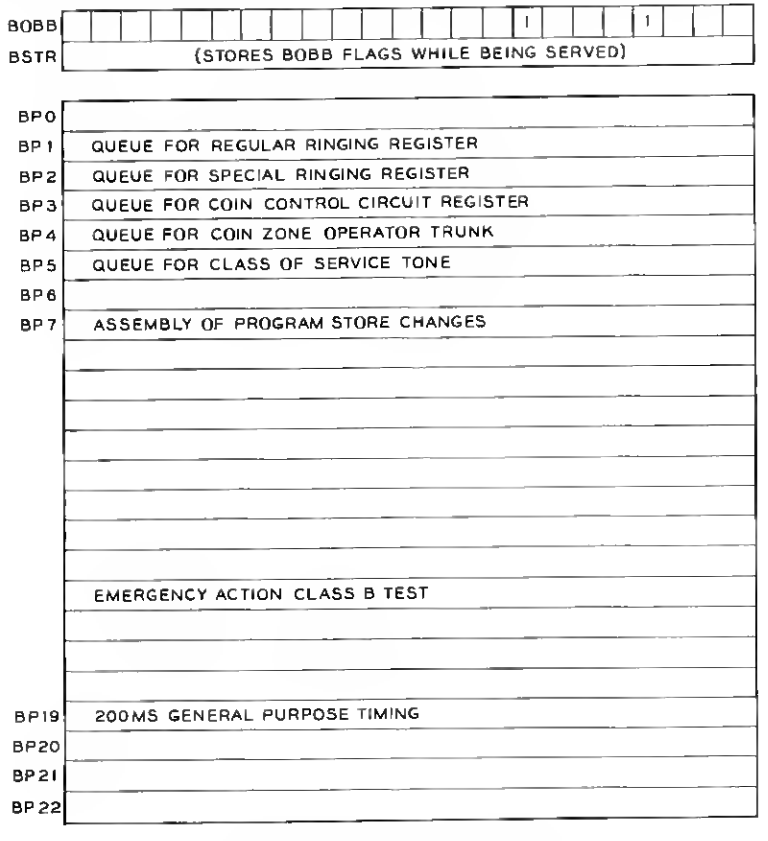

Fig. 10 — Class B ordered bits buffer.

figure, serves the coin control circuit register queue, so the flag would have been set to 1 by some program that needed such a call store register but found none available. At the same time, it would have left waiting on the corresponding queue whatever call register (perhaps an originating register) was already associated with the call. The program triggered by this flag from the ordered bits buffer will check to see if any coin control circuit registers have become available; if so, the one or more calls represented by registers on the queue will be served.

The other 1 in word BOBB of the ordered bits buffer, in bit 7, indicates that there are program store changes to be assembled preparatory to writing new cards for the translation area of the program store. This does not occur for days or weeks at a time, and the flag would have been set to 1 by a special teletypewriter message requesting this action.

The ordered bits buffer dispenser program begins by copying the contents of BOBB into the following word BSTR and zeroing BOBB. Then new flags can be set in BOBB even by interrupt-level programs while the 1's in BSTR are being served and erased from right to left, using the special TZRFZ instruction. (See Table III, step 5.)

4.3 *Interject Work*

The sixth class of base-level work is the interject class; it is not regularly scheduled at all in the ABACA ··· sequence. It has the highest priority of all, however, for a check is made after each task program in each of the other five classes for the existence of interject work, and if found it is done immediately. It is initiated by interrupt-level programs when they encounter work that cannot tolerate the delay it might suffer if put even in a class A buffer, but which should not be done in the interrupt program itself. The latter constraint might be that it is too lengthy for an interrupt level; or that it might interact harmfully with the interrupted base-level program, if both should try to change the contents of the same call store location. Therefore, it is scheduled to be done at the first natural break, by being interjected between task programs. The check for the presence of interject work is accomplished automatically as each task program returns control to the proper dispenser program by transferring to the standard program address "RETURN." At this point, the pair of instructions

MK	RETN	(move contents of "RETN" to index register K)
TKP	0, K	(if K's sign bit is positive, transfer to address in rest of K)

occur. "RETN" is the symbolic address of a call store word containing the dispenser program address to which control should be returned by each of its task programs; the first thing a task dispenser program does upon gaining control initially is to place this address in RETN. The sign bit of RETN is normally positive, but is made negative as a flag by any interrupt-level program wishing to interject a job. Thus the first two instructions of the program at RETURN read this call store word RETN and transfer to the address it contains, unless the sign bit has been made negative. In that case, program control passes to the succeeding orders, which form the interject control program. This program has certain special bookkeeping functions to perform, but essentially it transfers control to the one or more interjected jobs and then returns control to the dispenser address that was in RETN.

The interject priority class consists of a single dispenser program, the interject ordered bits buffer. At present only three kinds of jobs are definitely planned for this buffer, though more can be added if future requirements dictate. One is an emergency-action interject test, and another is the unloading of the hopper containing highest-priority reports from the network execution program. The third is the updating and distribution of information from the 100-millisecond timetable. For the first two, flags are set in the interject ordered bits buffer (and the master flag in RETN) when the relevant interrupt-level programs encounter the need either for the emergency test or the highest-priority network report. The flag for the 100-millisecond timetable is set by the 5-millisecond input-output low-priority timetable every twentieth entry.

The interject 100-millisecond timetable is central to all base-level timing. It is shown in Fig. 11, and will be described only briefly, because it is similar to the high-priority timetable for the H and J interrupt levels which has already been discussed in some detail in Section 3.1.3.

The supervisory line scan should be made every 100 milliseconds, nominally. There is no advantage in scanning excessively often, even in slack periods; routine maintenance checks and various administrative programs can usually make better use of the time. To prevent excessive scanning the line scanning program receives control from the class D ordered bits buffer when its flag bit is found to be 1. That flag is set once every 100 milliseconds from the interject 100-millisecond timetable. On the other hand, an extension of the 100-millisecond period, even if long enough to be noticeable, carries no serious penalty; in fact, since line scanning does take an appreciable amount of the processor's time, it is desirable that during overload proportionately less time be expended looking for work and more spent on work already in progress.

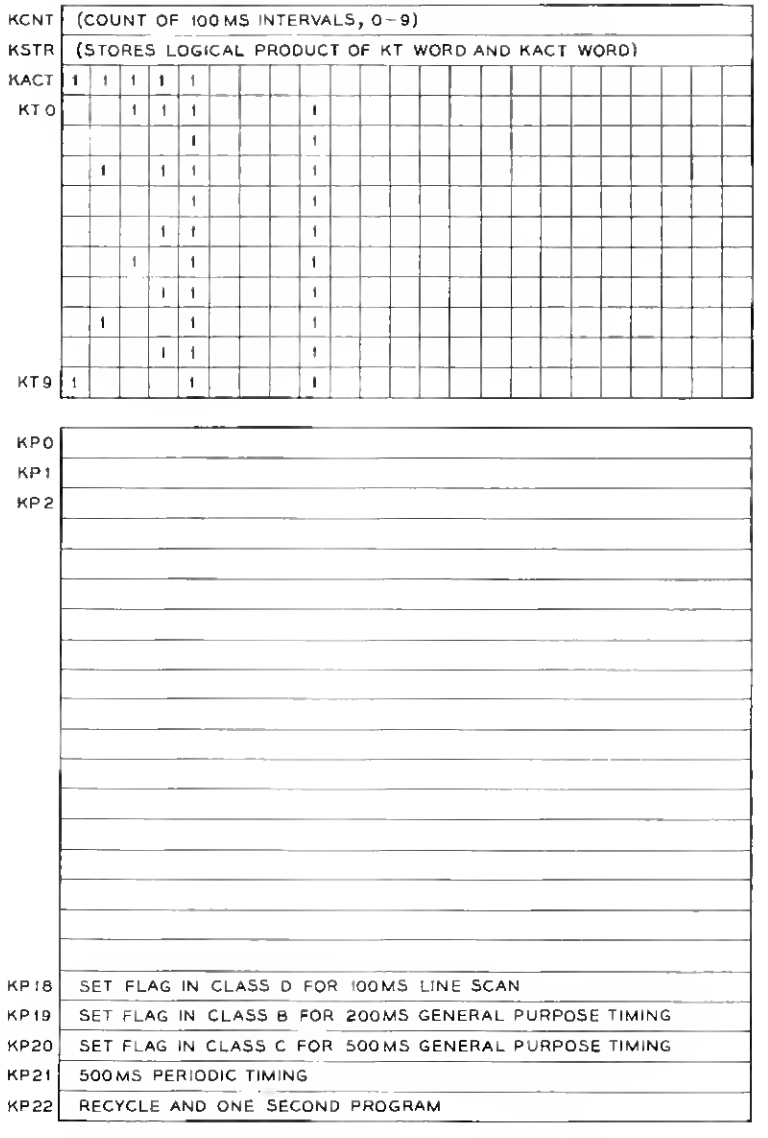

Fig. 11 — Interject 100-millisecond timetable.

The periods between executions of class D programs may exceed 100 milliseconds during periods of heavy traffic, and when they do the class D supervisory line scan flag will be set to 1 more often than it is served and made 0.

In column 19, 1's alternate with 0's, so the program whose address is

in row 19 of table KP0 will be entered every 200 milliseconds. All this program does is set a flag in the class B ordered bits buffer for 200-millisecond general-purpose timing, which will be discussed later.

All the programs entered from this table KP0 are, of course, executed as interject jobs, and as such must be kept short lest they themselves delay other interjeet work. The program whose address is in row 21, entered every 500 milliseconds, updates another timetable which contains 24 rows and thus is cycled through completely every 12 seconds. This timetable, however, has no associated table of transfer addresses; instead, after selecting the current row and ANDing it with the activity bit word, it ORs the result into the class C ordered bits buffer. (This divides the latter's bit positions into two kinds: those set by the 500-millisecond interjeet timetable; and those set by other means, like those of the class B ordered bits buffer already discussed. The ones set by other means render the corresponding columns of the 500-millisecond timetable useless, of course.)

4.4 *Timing*

As explained in earlier sections, the detection and discrimination of input signals is assigned to interrupt-level programs, and the resultant work of processing them to the base level. Somewhat analogously, time — in the form of a signal every 100 milliseconds — is an input to the base level, where it is detected and analyzed by interjeet programs, and the work that is found due is passed on to the lower base-level classes. When the intervals being timed become so long that the delays in performing base-level work are negligible by comparison, some of the timekeeping itself can be removed from the interject level. Thus class C has a 3-second timetable, and class E, 15-second and 15-minute timetables.

There remains an assortment of timing requirements not fulfilled by the timetables. Typically, certain kinds of call register programs may reach a point where a delay of a few hundred milliseconds or a number of seconds is required. One example is a TOUCH-TONE test trunk program, which must send out a failure signal 15 seconds after initiation of the test if a correct sequence of signals has not been keyed in by that time. General-purpose timing for such uses is provided by timing linked lists.

A hypothetical one-way 200-millisecond linked list is shown in use in Fig. 12. The only call store memory required for this list when not being used is the single word TIM200, sometimes called the "head cell"

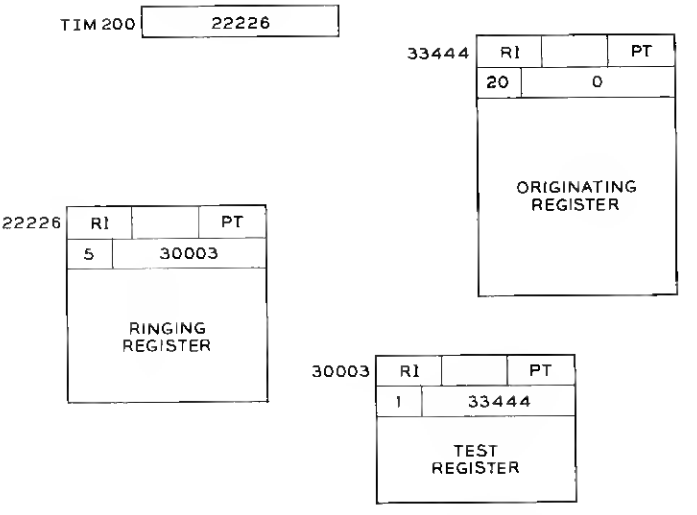

Fig. 12 — One-way 200-millisecond linked list for general-purpose timing.

of the linked list. When not in use, this word contains zero. In the figure, the ringing register was the last to be added of the three registers undergoing timing. The program which added it to the list found TIM200 containing 30003, the address of the test register. The ringing register was inserted in the list by having the "30003" stored in its second word, along with the "5" which specifies the number of 200-millisecond units of timing that are desired. Its own address, 22226, becomes the new contents of TIM200. The timing for whatever registers have been put on the list is administered by a task dispenser program given control every 200 milliseconds from the class B ordered bits buffer (Fig. 10). This program goes through the list, subtracting one from the count in the left part of the second word of each register. When the count is reduced to zero (as it will be next time for the test register, whose count is shown as "1"), the register is removed from the list (by replacing 30003 in the ringing register by 33444) and control is given to the appropriate task register program. The procedure by which this appropriate program is determined is used not only here, but in other situations in which a general program reports to a variety of other programs through their associated registers. Instead of a full program store address being stored in the call register, the RI and PT items in the register's first word are used. The RI ("register identification") selects a table of transfer addresses associated with the type of register,

and the PT ("program tag") is used to select from this table the address of the proper task program. (In this case, of course, that will be whatever program has been designed to take appropriate action when the test register times out.) This selection from the table, however, is made by altering the PT number slightly before using it as an index. This same alteration is performed by all programs reporting a certain class of information — including timing notification — to any call register. The report by some other program of a different class of information — e.g., a scan point change — would not make this alteration, and therefore could use the same PT number to select from the same table a different program. This means of distinguishing inputs eliminates many decisions within the task programs.

Note that to remove a register from a one-way linked list at an arbitrary time between scheduled inspections requires tracing through the list from the beginning. Since this could consume considerable time for a long list, two-way timing linked lists are also provided. Their use is quite similar, but each register must supply two words for the linking, one pointing forward to the next linked register and the other pointing back to the preceding. This means that any register in the chain can be directly dropped out, since it contains enough information for the linkage to be mended.

Most programs requiring general-purpose timing have an associated register to link; for those that do not, a common pool of timing registers is provided.

4.5 *Task Programs*

The variety of task programs precludes a comprehensive treatment within this paper, but at least some of this variety may be indicated. The word "program," when used to designate a subdivision of some larger program, is a static term referring to a set of instructions capable of performing a certain action or group of related actions when some or all of these instructions are executed in sequence. When control is given to such a program at one of its entry points, however, the sequential execution that follows typically runs through only part of the orders in that particular program, and then on into parts of other programs. It is this dynamic unit, consisting of the complete set of instructions executed in unbroken sequence, that is most significant in understanding the flow of control governed by the main program. "Task program" is often used in this dynamic sense, but "task execution" might be a less ambiguous term for the complete sequence of order executions from the

time a dispenser relinquishes control until control is returned to that dispenser. Thus, during a task execution only task programs are executed (apart from possible insertions of interrupt programs), but many and varied task programs may contribute to a single task execution.

There can be no "real-time gaps" within a task execution; such a gap necessarily signals the end of a task execution, and the suspended call or other function can regain control only through the main program mechanism. Task executions differ greatly in length, of course, depending on the amount of continuous work it is possible and desirable to do. As explained in earlier sections of this paper and in the paper on call processing,[2] the work that is done for one call at one time tends to be small, due to circuit limitations and timing requirements as well as the natural fragmentation of customer actions.

Task executions vary also in the number of different "static" task programs they involve. One well known but important programming technique that tends to use many programs in one task execution is the use of subroutines. In the No. 1 ESS program these range from short programs performing simple conversions of equipment addresses from one form to another, to complex network and translation routines which themselves use several levels of subroutines internally.

A subroutine, as the term is generally used, returns control to its client program when it has completed its work. Often, however, two or more task programs arrive at a point where the remaining work to be done in both or all cases is identical, and they can be joined at that point. There is no need to preserve the identity of the source merely in order to return control properly at the end of the task execution, because the standard transfer to RETURN, explained earlier, solves that problem.

The supervisory line scan is an example of a self-contained task program. It could be considered, of course, as two programs: one that repetitively compares the line scanner points with the line state bits, and another program or subroutine to which control is transferred when a proper match of these readings occurs that makes the entries in the service request hopper. If these two closely related programs are combined, however, they become a self-sufficient whole.

A contrasting example is provided by the task execution that begins with a digit just unloaded from a digit hopper. The initial task program is a part of the general call processing functional subdivision called "digit analysis, lines." It begins with the assumption that the K register contains the new digit in bits 21 through 18 (with the digit "0" registered in binary as "1010," corresponding to the ten pulses generated by a dial

telephone), and that the X register contains the address of the originating register associated with the call. This task program could have received control from either of two task dispenser programs, depending on whether the digit was produced by a dial or TOUCH-TONE telephone set. In handling the digit up to this point, the actions of the two task dispensers differ. The dialed digit task dispenser distinguishes the digit entry from the abandoned call entry it may also find in its hopper; the TOUCH-TONE digit task dispenser converts its hopper entry from a code indicating two active tone generators into a binary digit. From this point on, however, the distinction vanishes, and subsequent programs treat the digit in the same way regardless of its original form.

The initial decisions depend on the contents of the originating register whose address is in the X register. This register is a block of 16 consecutive call store words, assigned to a call as soon as the originating line is connected to a receiver and given dial tone, and retaining information for that call until a ringing connection is set up (for a local call) or until outpulsing is completed (for an outgoing call). The initial arrangement of information in the register is shown in simplified form in Fig. 13. (The type of information stored in some words or parts of words varies with the type of call, and can change as the call progresses.) The program first checks the single bit END in word 6 to see if dialing has already been interpreted as completed; if so (an unlikely event) the extra digit is ignored and control is immediately returned to the dispenser program by a transfer to RETURN. Assuming that dialing has not been previously completed, the program adds one to the digit counter (item DC in word 12), stores the new digit in its proper digit slot (DS1–DS10) as determined by the new value of the digit counter, and then checks to see whether that new value matches the number in item DCA in word 6. This number has been put there at some earlier point in the call to tell the present program whether additional decisions must now be made or whether nothing is required beyond the digit storage that has just been accomplished. If DC is not equal to DCA, control is returned to the dispenser program. If DC equals DCA, control is given to the program determined by item AADD in word 6. (The common program device used here to reduce item AADD from a full 18-bit call store address size is to transfer into a fixed table of transfer instructions, using the value of AADD to select the point in the table to which control is transferred.) The program receiving control could be any one of about thirty. (Since AADD is eight bits, the maximum table size is 256, of course.) For example, it might be the program analyzing the first digit, in which case DCA would have been set to

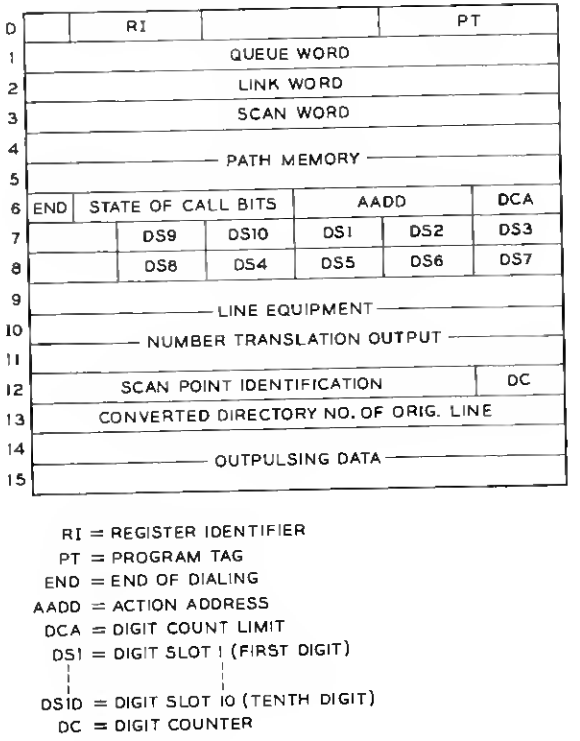

Fig. 13 — Simplified layout of originating register.

"1" by the program that connected the line to a receiver and dial tone. The analysis of the first digit involves checking to see whether it is a "1", "0", or the eleventh button on a TOUCH-TONE set, since any of these can have special significance. In an office without the "0" prefix option, an initial "0" necessarily completes dialing and requires a connection to an operator. In this case the task execution continues through subroutines in the translation and network area, transferring to RETURN only when it has made its first request for network action by placing proper entries in a peripheral output buffer. The foregoing assumes that no difficulties are encountered. If all operator trunks are busy, a request for a connection to overflow tone will be made instead; or, if no peripheral output buffer is available, the originating register will be left waiting in the corresponding queue before control is relinquished.

Referring again to the possible actions following a match of DC and

DCA, the program to which index AADD points might be one involved in providing a special service such as temporary transfer to a customer. The temporary transfer program would be controlling the call in this case, and would have seized an originating register to serve as a digit collector, placing the desired number of digits in DCA and a number in AADD that would lead to the appropriate temporary transfer task program.

Thus this digit analysis task execution may complete its work in a few order cycles, or it may branch into a multitude of different task programs before its sequence of instructions finally reaches a transfer to RETURN.

V. THE GENERIC PROGRAM

One of the important basic requirements stated earlier for the No. 1 ESS program is that it be generic — that is, that a single program be able to serve many offices with different features and characteristics, and during continuing growth. If the program does not change, then data to which it refers must change to take account of the different features, numbers of lines and trunks, types of signaling, and other variables. From a theoretic viewpoint, this concentration of all changeable information into one place instead of scattering it through the data and address fields of many instructions may seem a trifling difference. However, when a large number of offices are involved such an arrangement has substantial economic advantages in compiling programs, adding features, and making changes.

The constraint this places on the program design can be stated quite simply as the inability of the program to know anything that can change. Any quantity or address or option that can vary, as an office grows or from one office to another, must be looked up by the program in a concentrated data area of the program store. The call store may be used where faster reference is needed, but this may be considered an indirect reference to the program store, which must back up all long-term call store information. (In particular, when the office is first put into operation, the call store must be written to its proper initial state from program store data.)

The effect the generic program requirement has on memory allocation is essentially that, apart from the generic program itself, program store and call store are each divided into two parts. One area is of fixed size and fixed layout (though not fixed contents, of course), and the other

can be expanded and rearranged. A good call store example is provided by the hopper shown in Fig. 8. The four-word pointer block must be in the fixed layout part of call store, so the loading and unloading programs can refer to PBA, PBA+1, etc., in the address fields of their instructions. The hopper itself, however, could not be included in this area unless it were made big enough for the largest one needed in any office; but it is economical to reduce its size in smaller offices. The storage in the pointer block of the beginning and end addresses of the hopper itself allows the latter to grow or shrink as required in the expandable part of call store.

Similarly, call registers such as the originating and ringing registers are assigned in expandable call store, and while idle are chained together in linked lists, one for each register type. Only the two-word "head cell" of each linked list, containing the addresses of the first and last registers, is assigned in the fixed layout part of call store.

Nearly all call processing programs use such linked lists as their sole reference in seizing and releasing registers as needed. In program store, however, there must be a record in some form of the absolute addresses of all registers of each type. The call store initialization program, used when the office is turned on initially, must have such information; a few other programs also find it useful. This information is packed quite compactly, since reference to it is made only infrequently. Even so, the amount of program store space required for recording the locations of all registers in the biggest office is more than can be afforded in a small office. Hence the same division into fixed layout and expandable memory is made within the program store data area. A single word for each call register type is assigned at a certain address in the fixed layout portion. This word contains the address in the expandable part of program store of a variable-size block containing the locations of all registers of that type.

VI. SUMMARY

The reliable and prompt switching functions required of No. 1 ESS put stringent requirements on the program organization. The types of equipment with which the program must work and the services that must be provided are large in number. For a wide range of office sizes and traffic volumes, economy must be stressed, both for the initial installation and for growth. The program organization outlined was chosen because it satisfies these requirements. Certainly other means of implementation are possible.

The interrupt system helps to guarantee prompt attention to tasks that require it. To satisfy the input-output tolerances, it is also necessary to limit the time during which a given task is continuously processed. Most call programs have natural break points, and others must be created to be compatible with the equipment. Lengthy maintenance and administration programs are arbitrarily divided into segments whose execution time does not exceed a suitable limit.

This fragmentation of work, in turn, requires a system of call store registers and buffers in which to store data until processing can be resumed. These and other memory units are designed for ease of growth and economy throughout the range of office sizes.

Both the input-output and base level main programs follow simple schedules which check for work to be done according to its urgency, without consuming an inordinate amount of time.

To increase the efficiency of the very frequently executed input-output programs, special orders were devised, and certain groups of these orders are repeated in program store to avoid the extra transfer time that program loops would cost. For the greater part of the program, however, a very general order structure is used, which should be equally well suited to functions not yet conceived. For this majority of programs, furthermore, subroutines are heavily used to reduce program storage space. This organization also makes it easier to add new features to the program, since use can be made of existing program blocks and subroutines.

However, a major aim has been to minimize the causes that require an addition or any other change to the program itself. This is done by effecting a separation of the program proper from the parameters associated with the features of a particular office and the variables of growth.

VII. ACKNOWLEDGMENTS

The program organization of No. 1 ESS is the product of many contributors, from the areas of system engineering and design as well as call and maintenance programming. The choice of program techniques was closely coordinated with the analysis of the actions required for call processing, and depended also on the instructions, both general and special purpose, provided by the central control design. In particular, we wish to acknowledge the contributions of S. Silber, who provided many of the basic ideas for the main programs and is responsible for their actual design and programming.

REFERENCES

1. Keister, W., Ketchledge, R. W., and Vaughan, H. E., No. 1 ESS System Organization and Objectives, B.S.T.J., this issue, p. 1831.
2. Drew, G. G., Carbaugh, D. H., Ghiron, H., and Hoover, Mrs. E. S., No. 1 ESS Call Processing, B.S.T.J., this issue, Part 2.
3. Harr, J. A., Taylor, F. F., and Ulrich, W., Organization of No. 1 ESS Central Processor, B.S.T.J., this issue, p. 1845.
4. Downing, R. W., Nowak, J. S., Tuomenoksa, L. S., No. 1 ESS Maintenance Plan, B.S.T.J., this issue, p. 1961.
5. Martellotto, N. A., Oehring, H., and Paull, M. C., PROCESS III, A Compiler-Assembler for No. 1 ESS, B.S.T.J., this issue, Part 2.